Recitation on Sorting and Searching

Dr. S. Antoun

Task 1: Sorting Arrays

Write a program that creates three identical arrays, `list1`, `list2`, and `list3` of 5000 elements. The program then sorts `list1` using bubble sort, `list2` using selection sort, and `list3` using insertion sort and outputs the number of comparisons and item assignments made by each sorting algorithm.

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

// Function prototypes
void bubbleSort(int arr[], int n);
void selectionSort(int arr[], int n);
void insertionSort(int arr[], int n);

// Function definitions

// Bubble Sort
void bubbleSort(int arr[], int n) {
    int comparisons = 0;
    int assignments = 0;
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            ++comparisons;
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                ++assignments;
            }
        }
    }
    std::cout << "Bubble Sort Comparisons: " << comparisons << std::endl;
    std::cout << "Bubble Sort Assignments: " << assignments << std::endl;
}

// Selection Sort
void selectionSort(int arr[], int n) {
    int comparisons = 0;
    int assignments = 0;
    for (int i = 0; i < n - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < n; ++j) {
            ++comparisons;
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
        ++assignments;
    }
    std::cout << "Selection Sort Comparisons: " << comparisons << std::endl;
    std::cout << "Selection Sort Assignments: " << assignments << std::endl;
}

// Insertion Sort
void insertionSort(int arr[], int n) {
    int comparisons = 0;
    int assignments = 0;
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;
        ++assignments;
        while (j >= 0 && arr[j] > key) {
            ++comparisons;
            arr[j + 1] = arr[j];
            --j;
            ++assignments;
        }
        arr[j + 1] = key;
        ++assignments;
    }
    std::cout << "Insertion Sort Comparisons: " << comparisons << std::endl;
    std::cout << "Insertion Sort Assignments: " << assignments << std::endl;
}

int main() {
    const int size = 5000;
    int list1[size], list2[size], list3[size];

    // Filling arrays with random numbers
    srand(time(0));
    for (int i = 0; i < size; ++i) {
        list1[i] = rand() % 10000; // Generating random numbers between 0 and 9999
        list2[i] = list1[i];
        list3[i] = list1[i];
    }

    // Sorting and outputting comparisons and assignments
    bubbleSort(list1, size);
    selectionSort(list2, size);
    insertionSort(list3, size);

    return 0;
}
```

Task 2: Insertion Function

Write a function, `insertAt`, that takes four parameters: an array of integers, the number of elements in the array, an integer (`insertItem`), and an integer (`index`). The function should insert `insertItem` in the array at the position specified by `index`. If `index` is out of range, output an appropriate message. (Note that `index` must be between 0 and the number of elements in the array; that is, 0 ¡= `index` ¡ the number of elements in the array.) Assume that the array is unsorted.

```cpp
#include <iostream>

// Function prototype
void insertAt(int arr[], int& size, int insertItem, int index);

// Function definition
void insertAt(int arr[], int& size, int insertItem, int index) {
    if (index < 0 || index > size) {
        std::cout << "Index out of range!" << std::endl;
        return;
    }
    for (int i = size; i > index; --i) {
        arr[i] = arr[i - 1];
    }
    arr[index] = insertItem;
    ++size;
}

int main() {
    const int maxSize = 100;
    int arr[maxSize] = {1, 2, 3, 4, 5};
    int size = 5;

    // Test the insertAt function
    insertAt(arr, size, 10, 2);

    // Output the array after insertion
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Additional Task: Binary Search

Write a function named `binarySearch` that takes an array of integers, the number of elements in the array, and a target integer to search for. The function should implement the binary search algorithm to find the target integer within the array. If the target integer is found, the function should return its index; otherwise, it should return -1 to indicate that the target is not in the array. Assume that the array is sorted in ascending order.

Here's the function signature:

```cpp
int binarySearch(int arr[], int size, int target);
```

The task is to implement the `binarySearch` function and demonstrate its usage in a program that creates an array of integers, prompts the user for a target integer, and then searches for that integer using the `binarySearch` function.

```cpp
#include <iostream>

// Binary Search function
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid;
        }
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Target not found
}

int main() {
    const int size = 10;
    int arr[size] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
    int target;
    std::cout << "Enter the number you want to search for: ";
    std::cin >> target;
    int result = binarySearch(arr, size, target);
    if (result != -1) {
        std::cout << "The target " << target << " is found at index "
        << result << std::endl;
    } else {
        std::cout << "The target " << target << " is not found in the array."
        << std::endl;
    }
    return 0;
}
```